TARGETGUARD: FRESHNESS-CERTIFIED TARGET INTEGRITY FOR STATE-CHANGING LLM ACTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

An API call can succeed while modifying the wrong entity. TargetGuard makes this distinction explicit through a target-integrity benchmark contract and a freshness-certified selective execution gate. Given a proposed action and a public typed relation, the gate admits a unique evidenced target only after revalidating its relation, pre-state, and executor-version certificate; otherwise it abstains. We establish conditional target integrity when the relation links candidates to the requested entity and the executor protects validation and mutation atomically. Observationally indistinguishable targets require abstention or additional evidence.

Our primary comparison uses 192 controlled Git and SQLite cases. Raw dispatch records 96 wrong-target writes. Both TargetGuard and an independently implemented typed revalidation reference record none, retain all 96 uniquely identifiable operations, and abstain on all 96 ambiguous cases, without additional LLM calls. Both reject all 96 actions made stale before revalidation in a separate stratum. These tests measure stale-state detection, not protection against concurrent changes between validation and mutation. Forty-two application replay cases and an eight-case resolver boundary audit test adapter behavior. A controlled, oracle-assisted 12-pair intervention isolates target substitution under fixed model responses. A separate 1,800-case analytical suite compares policy decisions, not persisted writes.

The result is a reusable contract for measuring correct-target execution and the valid work retained by execution policies. It does not establish an advantage over typed revalidation or estimate deployment failure prevalence.

1 INTRODUCTION

An LLM agent can issue a tool call that is valid in every syntactic and permission-level sense, yet mutate the wrong entity. A file identifier can exist, be authorized, and pass the API schema while referring to the other "contract.docx" in a shared workspace. An API return value can therefore be distinct from the post-state target-integrity audit. We call this a target-integrity failure and report it as an independent evaluation dimension; the archived records do not establish a per-case API/task-score mismatch rate.

TargetGuard is a selective execution algorithm for this failure. It converts an identifier-bearing state-changing LLM action into one of two outcomes: either a uniquely verified dispatch, or a pre-mutation abstention. The algorithm receives a typed target, a runtime-visible relation, and an atomic executor. It changes only the target field and makes no additional LLM call. Thus it does not ask the model to guess better; it decides whether the current evidence is sufficient to permit a side effect. This is the paper's single question: when may an agent action safely cross the boundary from language to persistent state?

The primary executor result is one target-integrity comparison over 192 unique-or-ambiguous Git and SQLite cases. Raw dispatch makes 96 wrong-target writes. Both an independently implemented typed revalidation reference and TargetGuard make none, retain all 96 identifiable actions, and abstain on all 96 ambiguous actions. In a separate 96-case freshness stratum, both reject every action made stale before revalidation. This parity is informative: the contribution is not a claim that certification outperforms a competent reference monitor, but a portable contract that makes the same

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

safety boundary explicit and auditable. Analytical policy tests, 42 application replays, and a controlled fixed-response intervention provide complementary checks; their outcomes are not pooled with the primary executor result.

Running executor example. In a native SQLite case, the public alias `ticket_01` returns two type-compatible candidates, `entity:1001` and `entity:2001`. The frozen task contract designates `entity:2001` for a single `close_ticket` mutation. Raw dispatch commits the mutation on `entity:1001`; the API-level operation succeeds, but the persisted state leaves the designated entity unchanged. TargetGuard observes the same ambiguous public relation, abstains before the mutation call, and leaves both entities unchanged. This controlled case illustrates the measured quantity; it does not estimate how often such ambiguity occurs in deployment.

Stateful benchmarks already place agents across heterogeneous tools and services (AppWorld and ToolSandbox, discussed below). A wrong-target write can require recovery even when the API accepts the call. Its consequence depends on the application; we do not estimate its economic cost. TargetGuard studies the boundary between model-produced actions and externally visible state, not open-world entity resolution or general agent capability.

The distinction is between checking a valid identifier and establishing its relation to the requested entity. Syntax checks alone do not establish that relation. Database constraints, knowledge graphs, and application logic can help establish it and can implement the same safeguards. TargetGuard makes their obligations explicit: evidence must identify a unique compatible target and remain valid at commit time. The contract allows different implementations to be evaluated against the same target-integrity outcomes.

The paper makes three focused contributions. First, it defines target integrity as an explicit evaluation dimension for identifier-bearing state-changing actions and gives a machine-readable benchmark contract with one designated mutation and post-state audit per row. Second, it gives a selective execution rule whose commit-time certificate binds a unique typed target to the public relation, pre-state, and executor version, and proves relative target integrity and abstention necessity. Third, it evaluates fixed reference policies, executors, and schemas under one auditable safety–utility comparison. The strong typed revalidation reference matches TargetGuard on the fixed native cases; TargetGuard's empirical role is therefore a benchmark reference implementation with an explicit certificate and adapter interface, rather than a claim of resolver superiority. Model-backed runs are retained as compatibility boundaries rather than pooled learned-resolver claims.

2 Related work and novelty boundary

Recent work has studied wrong-entity actions, resolution preconditions, confidence gating, provenance, and binding drift in tool-augmented agents (see Entity Binding Failures in Tool-Augmented Agents and Binding Drift in Multi-Step Tool-Augmented Agents). The first of these is the closest conceptual prior: it already isolates right-tool/wrong-entity failures and evaluates preconditions, confidence gating, clarification, and provenance at substantially larger scale. The second studies multi-step drift and an LLM re-verifier. We therefore do not claim novelty for discovering entity-binding failures, for confidence gating, or for provenance. Our narrower object is complementary: a commit-time certificate that binds a resolved target to the public relation, pre-state, executor version, and one auditable mutation. TargetGuard revalidates that certificate immediately before dispatch and otherwise abstains; it never silently substitutes a newly ranked candidate. The paired fixed-response design identifies the effect of this execution-side target-only projection with no additional LLM call. It is not a replacement for the broader entity-resolution evaluations in these works. Stateful tool-use benchmarks such as AppWorld, ToolSandbox, and τ-bench motivate state-based evaluation of multi-step API interaction. We specialize that setting to a different question: whether a valid mutation can be redirected to the wrong cross-application entity, and whether a target-only intervention can prevent that side effect. Recent adjacent benchmarks make the distinction important rather than eliminating it: E-Bench evaluates 323 multi-step, state-changing tasks with database-diff grading (E-Bench), while EnvTrustBench evaluates whether agents ground actions in reliable environment evidence across scaffolds and models (EnvTrustBench). These are complementary capability and evidence-grounding benchmarks. We add a target-integrity scorecard alongside these broader task and evidence-grounding scores: intended mutation, wrong-target write, safe abstention, and identi-

fiable false rejection under one public typed relation. We therefore claim an additional measurement axis, not a first state-changing-agent benchmark or that existing benchmarks cannot detect mistargeted state changes. Our narrower contribution is a target-only execution contract and conditional invariant, instantiated by an auditable cross-application protocol. The deterministic path uses zero additional LLM calls, while lookup and wrapper overhead remain explicit. The learned comparator is a same-condition diagnostic, not a superiority claim.

Work on auditing tool-agent evaluation and runtime mitigation likewise argues that stage-level errors can be hidden by a single end-to-end score and that rejection and recovery should be measured separately (AgentProp-Bench). This supports our row-level mutation audit, but it is not an empirical comparison: our paper does not reproduce its monitor or claim superiority over runtime verification. Verifiably Safe Tool Use for LLM Agents derives safety requirements through system-theoretic process analysis and proposes capability-enhanced MCP constraints (Doshi et al.). Our contract is narrower: it concerns identity-preserving target substitution, not general authorization or information-flow control.

3 Problem and Safety Boundary

Let e be the requested real-world entity and $id_a(e)$ its identifier in application a. An agent emits a mutating call $m(id_b)$. An alias perturbation replaces the intended target with a valid identifier associated with another relation. The repair operator R may replace only the target identifier using runtime-observable relation or provenance information; all other arguments remain unchanged. The no-additional-call constraint requires R to issue no additional LLM request.

Definition 1 (Typed public alias resolution)**.** *For a mutating call m with target field p, let $x = (r, \tau, \rho, O)$ contain the requested alias r, target type τ, optional provenance ρ, and public observations O. An adapter constructs the candidate set*

$$C(x) = \{c : \ (c, r', \tau', \rho') \in O, \ r' \equiv r, \ \tau' = \tau, \ (\rho = \varnothing \ \vee \rho' = \rho)\},$$

where $\equiv$ is whitespace/case normalization only. Define the auditable evidence predicate

$$E(c, x) = \mathrm{valid}(O) \wedge \mathrm{type}(c) = \tau \wedge (\rho = \varnothing \vee \mathrm{prov}(c) = \rho) \wedge \mathrm{bound}(p, \tau),$$

where $\mathrm{valid}(O)$ means that the complete relation is present and well formed, and $\mathrm{bound}(p, \tau)$ means that the adapter declares the action's target field p to accept stable identifiers of type τ. Typed alias resolution is the selective decision rule

$$R(x) = \begin{cases} \textsc{repair}(c), & C(x) = \{c\} \text{ and } E(c, x), \\ \textsc{reject}, & |C(x)| \neq 1 \text{ or } \neg E(c, x). \end{cases}$$

An explicit stable identifier may satisfy E only when it is unique and type/provenance-consistent. Thus the rule never resolves a tie by ranking or guessing.

Theorem 1 (Relative target integrity under atomic projection)**.** *Assume that (a) the resolver receives only the public runtime input x and no hidden intended-target label; (b) the adapter's typed relation is sound, so every candidate admitted by R denotes the requested entity; (c) the wrapper changes only the target field p; and (d) the executor protects certificate revalidation and mutation in one atomic boundary. Then every dispatched repaired call targets the requested entity. If $C(x)$ is not a singleton, or if $C(x) = \{c\}$ and $E(c, x)$ fails, $R(x) = \textsc{reject}$ and the wrapper performs no mutation.*

Proof. For an admitted case, (b) gives that the unique substituted identifier c denotes the requested entity. By (c), every non-target argument and the operation itself remain unchanged. By (d), relevant state is protected between certificate validation and the mutation, which therefore targets c. For a non-unique or under-evidenced case, the second branch of R returns REJECT; (d) then implies zero mutation and therefore zero wrong-target write. Relation soundness is an explicit adapter obligation, whereas unique selection, target-only projection, and fail-closed commit are algorithm guarantees; the theorem does not assert that an arbitrary deployment exposes a sound relation. □

This is a conditional contract theorem. Assumption (b) supplies the semantic link between a public candidate and the requested entity; uniqueness alone does not establish user intent. Relation

completeness requires a trusted adapter or a completeness witness; well-formed records alone cannot establish it. Type, provenance, freshness, and state-diff checks test the remaining obligations. Benchmark construction supplies the offline intended target used to test the adapter's semantic obligation.

Theorem 2 (Necessity of abstention under observational equivalence)**.** *Let two possible task worlds expose the same runtime input x to a resolver, including the same candidate identifiers, aliases, types, provenance, and operation, while the intended target is c_1 in the first world and $c_2 \neq c_1$ in the second. Any deterministic resolver that dispatches a candidate on x selects the same candidate in both worlds and therefore makes a wrong-target write in at least one world. Thus a zero-wrong-target policy over both worlds must reject or obtain additional distinguishing evidence. The same conclusion holds for a zero-error randomized resolver by conditioning on each random seed.*

Proof. Because the runtime inputs are identical, determinism makes the resolver's output identical in the two worlds. If that output is a dispatch to c_1, it is wrong in the second world; if it is a dispatch to c_2, it is wrong in the first; and any other candidate is wrong in both. Thus only rejection, or new evidence that makes the worlds distinguishable, avoids a wrong dispatch. □

Algorithm 1: selective target-integrity resolver. For a proposed mutating action m with target field p, runtime input x, and adapter A, the algorithm is the partial action transformer

$$\mathrm{STI}(m, p, x, A) = \begin{cases} \mathrm{dispatch}(m[p \leftarrow c]), & C = A(x) = \{c\} \ \wedge \ E(c, x), \\ \mathrm{abstain}, & \text{otherwise.} \end{cases}$$

The first branch is the only branch that can mutate state; abstention is a first-class output rather than an execution error. The algorithm therefore has a directly testable safety contract: the adapter must expose the public relation, the evidence predicate must be auditable, and validation and dispatch must be atomic. These are adapter and executor obligations, not guarantees provided by language-model output.

For a permitted dispatch, STI also issues a certificate

$$\kappa = (c, h_R, h_I, \nu, v),$$

where c is the selected stable identifier, h_R is a canonical digest of the complete public relation and reference, h_I is a digest of the pre-mutation audit snapshot, ν is the executor version, and v is the TargetGuard algorithm version. The adapter must revalidate κ immediately before the atomic call: the relation, reference, pre-state, and executor version must match, and rerunning the same selective rule must return c. Otherwise the only legal outcome is ABSTAIN; a stale certificate is never repaired by choosing a new target.

Theorem 3 (Certificate revalidation safety)**.** *Assume the certificate covers all target-relevant state, canonical digest equality implies equality of the checked values, and the executor protects revalidation and mutation in one atomic boundary. If revalidation confirms the certified relation R, pre-state I, executor version, and selected target, dispatch is target-consistent with the certificate. A mismatch at revalidation causes rejection before mutation.*

Proof. The certificate commits to c, the canonical relation digest, the pre-state digest, and the executor version. Under the digest assumption, matching digests and repeated unique resolution identify c under the same evidence. Atomic protection preserves that evidence until mutation. A mismatch fails revalidation and prevents dispatch. With cryptographic hashes the equality assumption excludes collisions; the certificate checks current values, not every historical change that may have been reverted. □

The wrapper is not a second language model: it builds $C = A(O, m, p)$, substitutes only the unique evidenced identifier, and otherwise returns REJECT before dispatch. With an index keyed by normalized alias, type, and provenance, construction costs $O(|O|)$ and a query costs $O(k)$ for k matches (or $O(|O|)$ without an index); relation storage is $O(|O|)$. The certificate adds relation and state hashing plus a pre-dispatch revalidation pass, whose cost depends on the serialized evidence size; $O(k)$ is not the end-to-end cost. The observational-equivalence theorem explains why rejection is necessary rather than merely conservative. The adapter is therefore an implementation of an explicit interface, not the algorithm itself: changing the application changes A and the relation schema, while the decision rule, proof obligation, and audit invariant remain unchanged.

Mutation-risk objective. We use $J_\lambda(\pi) = \lambda_w P(W) + \lambda_f P(F) + \lambda_c E[C] + \lambda_t E[T]$, where W is a wrong-target write, F a false reject on an identifiable case, C collateral mutation, and T dispatch cost. Rejection of an unidentifiable collision is reported as safe abstention, not F. The resolver normalizes aliases, filters by type/provenance, admits a unique evidenced candidate, and rejects ties or under-evidenced high-risk mutations; hence a dispatched unique-verifiable case cannot target another candidate.

Runtime boundary. The D12 fixture names an intended entity; the model is not scored as an open-world resolver. Repair-off substitutes the registered collision target, while repair-on applies the offline-designated projection. This oracle-assisted mechanism experiment is distinct from public-only runtime resolution. The adapter is the explicit portability boundary: the wrapper does not infer an authoritative relation from hidden evaluator state. With an indexed public relation, index construction is linear in the observed records and target-field substitution is constant-time. Lookup, hashing, revalidation, and auditing still have costs despite zero additional model calls. The conditional invariant above follows for any adapter satisfying the stated relation and atomic-dispatch assumptions. Our experiments instantiate this operator with an authoritative benchmark relation; they do not establish that an arbitrary adapter can resolve open-world aliases.

Transfer proposition. The safety result is portable at the contract level, not as an untested application claim. For any typed target field, authoritative relation adapter, and atomic wrapper satisfying the invariant, replacing the adapter leaves the projection, rejection rule, and proof obligations unchanged. A new application must expose the relation, target-field mapping, ambiguity rule, and atomic dispatch test; otherwise the output is "not identified" or REJECT. This is a formal portability claim, not empirical open-world generalization.

Benchmark instantiation and database neutrality. An admissible fixture is a tuple (I, O, m, p, R, G): an initial state I, public observations O, one mutating call m, its typed target field p, a public relation R used by the adapter, and an offline gold record G containing the intended outcome and any collision construction. A task generator must freeze these fields before execution, emit an exact-one-mutation executor, and retain the complete row-level audit, including rejected and failed rows. The primary causal track has no train/dev/test split because it learns no parameters; its split unit is the preregistered fixture list. A learned comparator must instead declare a task-disjoint train/dev/test split and report clean retention and collision recall separately from mutation utility.

Adapter contract and fixed reference policies. Each adapter manifest additionally names the application family, schema and executor versions, target-field mapping, relation digest, and atomic before/after audit implementation. A scored row is admissible only if it contains the alias or stable identifier, target type, optional provenance, public candidate relation, pre-state snapshot, exactly one designated mutation, and post-state diff; the offline label is retained for scoring but is never runtime input. The benchmark reference set is raw dispatch, static barrier, alias-only matching, alias-plus-type matching, a typed transactional reference, and an offline oracle. TargetGuard is the proposed certified policy. The oracle uses construction-only information and is not deployable. Provenance-only gating is an admissible adapter-specific extension, but is not silently substituted for a populated primary frontier arm. The typed transactional policy is our internally implemented strong reference policy, not a claim about a published named baseline; its checks and visible inputs are specified per experiment below. This makes a new database or API an adapter contribution rather than a new algorithmic variant. The complete machine-readable field list and extension rules are included in the supplementary artifact.

Our CPU contract uses an application/schema-disjoint split: calendar and shopping train, CRM development, music and tasks held out. Its 25 symbolic cases test resolver semantics, not model generalization, and separate wrong-target decisions, identifiable false rejects, and safe abstention. A future learned resolver must use this split and report the same Pareto coordinates.

As a separate adapter check, we map the frozen H2H public relations into the same interface across four endpoint shapes and two application families. The CPU audit resolves all four clean and all four controlled-collision cases, and abstains on all four ambiguous cases, with zero wrong-target decisions and zero identifiable false rejects. This validates adapter compatibility, not model utility or cross-application deployment performance.

The protocol is database-agnostic rather than database-free. A MySQL, SQLite, or service-backed implementation is admissible if it exposes a read-only public relation, a target-field adapter, an ambiguity rule, and an atomic mutation with a before/after audit. The database itself is not an oracle: G may construct and score a counterfactual offline, but a runtime repair must not query hidden task state or a gold table. We therefore release a candidate protocol and leaderboard schema—task utility, wrong-target writes per 1,000 mutations, collateral mutation, false rejection, errors, and model/wrapper cost—rather than claiming a populated cross-model leaderboard. New applications and models can be added only by supplying the same adapter contract and reporting the same fields. The machine-readable protocol and adapter specification are included in the supplementary artifact and are versioned there independently of the paper.

The CPU resolver test applies this rule to five unrelated domains and is semantic contract evidence, not model utility or deployment generalization.

To test scale without conflating deterministic policy behavior with model behavior, the released stress generator expands the same contract to 1,800 rows across nine application namespaces and nine disjoint schemas. It fixes eight regimes: unique provenance, explicit stable identifier, typed collision filtered by type, normalized unique alias, ambiguous collision, missing provenance, provenance mismatch, and unsupported alias. The application-level split is 600/400/800 train/dev/test rows, with no application or schema crossing a split. The result has 900 identifiable-case successes, 675 safe abstentions on unidentifiable cases, zero wrong-target decisions, and zero identifiable false rejects under its frozen scenario labels. Since this suite is a deterministic policy stress test, these are analytical scenario outcomes rather than executor-observed persisted writes; it is reported separately from the model utility tables.

4 A SHARED CONTRACT, SEPARATE EVIDENCE LAYERS

The execution contract specifies the path:

$$\text{proposed action} \rightarrow \text{typed public relation} \rightarrow \text{certificate} \rightarrow \{\text{commit}, \text{abstain}\} \rightarrow \text{state-diff audit}.$$

Resolver-only tests stop before mutation, and analytical comparisons assign scenario outcomes rather than execute this path. TargetGuard makes the relation, pre-state, and executor-version obligations explicit and revalidates them at commit time. The offline oracle is an upper bound, not a deployable input. The main executor table compares raw dispatch, the independent typed revalidation reference, and TargetGuard on identical deterministic fixtures. The supplementary six-policy analytical comparison predates that executor add-on and leaves typed transactional unevaluated rather than imputing its result. Consequently each comparison has a declared scorecard: intended mutation, wrong-target write or scenario label, false rejection, safe abstention, unsupported input, collateral mutation, and cost when available.

The evidence is deliberately nested rather than pooled. The 1,800-case deterministic suite measures an analytical challenge-policy comparison. It compares six fixed policies; it is not a continuous threshold sweep. Forty-two exact-one ToolSandbox and AppWorld Filesystem replays test two application adapters. A preregistered 288-case challenge then exercises eight mutation schemas on native Git and SQLite executors, each policy execution receiving fresh state; its 192-case primary stratum supplies the executor comparison, while its 96-case freshness stratum tests detection of a state change introduced before revalidation. Finally, the D12 paired intervention holds the model response fixed and changes only the certified target supplied to the executor. These layers test policy, execution, portability, and mechanism along one chain; their rates are never pooled.

5 EVALUATION PROTOCOL

Fixtures specify inputs before execution; rejected and failed records are retained. Executor results combine mutation records with state-diff checks; analytical results use scenario labels. Runtime resolution sees the proposed alias or identifier, type, optional provenance, and public candidate relation; the intended target remains in an offline ledger used only for construction and scoring. A row is admissible only if the executor performs zero mutations before the decision and exactly one designated mutation after an admitted dispatch. AppWorld's simulator clock is not treated as

wall-clock latency; model duration, token metadata, call counts, deterministic lookup time, and executor/audit time are reported in their respective tracks.

The independent-executor challenge freezes four Git mutations (append, replace, delete, and rename) and four SQLite transactions (close, cancel, assign, and archive). For each schema, 12 deterministic instances are run under unique, ambiguous, and stale conditions. The frozen raw/TargetGuard run contains 288 case keys and 576 policy executions; a CPU-only add-on replays an independent typed revalidation policy on the same 288 keys. All 288 fixture digests match the corresponding frozen raw rows. Unique and ambiguous cases form the primary comparison; stale cases are a separate freshness audit. The balanced composition is diagnostic and does not estimate deployment prevalence. All three policies use the same native mutation and post-state audit. Raw dispatch takes the first alias match. The typed revalidation reference normalizes and filters the typed relation, requires a unique candidate, and re-reads the relation and pre-state immediately before mutation. TargetGuard binds the same obligations into a revalidated certificate. The experiment therefore compares complete execution policies under common fixtures and auditing; it does not assume that certification must beat a well-built monitor. The stale stratum injects the external change before this revalidation. It therefore tests stale-certificate detection, not an adversarial interleaving between validation and mutation. The theorem's atomicity conclusion applies only when an executor enforces the assumed lock, compare-and-swap, or transaction boundary across validation and commit.

Native scoring and retained evidence. The runner compares the executor-reported mutation target with the fixed intended target; Git diffs and SQLite snapshot differences check changed objects and collateral effects. This is not a fully independent final-state scorer: records retain digests and outcome diagnostics rather than all original snapshots. All native runs report zero runtime errors. The runner's generic non-dispatch labels do not independently certify error handling; we do not extend the observed safe-abstention counts to untested exception paths.

For D12, AppWorld 0.1.3 and data 0.1.0 are locked across six Spotify and six Phone+Venmo tasks. Spotify exercises numeric identifiers and Phone+Venmo email identifiers. The preregistered paired estimand is $\Delta_{\text{pair}} = \Pr(W^{\text{off}} = 1, W^{\text{on}} = 0) - \Pr(W^{\text{off}} = 0, W^{\text{on}} = 1)$, where W denotes a wrong-target write. Both strata have the same direction: 6/6 wrong writes with raw dispatch and 6/6 designated mutations with target-only projection. The result is oracle-assisted mechanism evidence because the collision and intended target are fixed offline; deployment-time TargetGuard must instead certify a unique target from the public relation. All broader model-compatibility runs, failed screens, and stop-rule outcomes are retained in the supplement and are not pooled into the headline comparison.

6 One target-integrity comparison across executors

The experiments support one claim at one level: a target-integrity contract can preserve an identifiable mutation and refuse an unresolved one before it changes state. The following results measure that boundary separately from model grounding, natural-workload prevalence, and unrestricted agent utility.

Table 1: Primary target-integrity comparison on 192 unique/ambiguous native Git and SQLite cases. The typed revalidation add-on uses the same case keys and matches all 288 fixture digests. Outcomes use executor target reports and state-diff checks as described in Section 5. Counts are mutually exclusive within rows. WTM/1k is wrong-target mutations per 1,000 attempts and ID retain is retention on uniquely identifiable rows.

Policy	Intended	Wrong	False reject	Safe abstain	WTM/1k	ID retain
Raw dispatch	96	96	0	0	500	100%
Typed revalidation	96	0	0	96	0	100%
TargetGuard	96	0	0	96	0	100%

The primary headline is 500 to 0 observed wrong-target writes per 1,000 attempts in the independent-executor comparison, without sacrificing identifiable work: all three policies retain 96/96 unique rows. The 96 ambiguous rows are safe abstentions for typed revalidation and TargetGuard, but

Table 2: Schema-level decomposition of the 192-case primary executor comparison. Each cell is out of 12 controlled replicates. "Unique" is an intended commit; "ambiguous" is a wrong-target write for raw dispatch and a pre-mutation abstention for TargetGuard. Typed revalidation matches the TargetGuard columns in all eight schemas and is omitted here only to keep the decomposition legible. The repeated rows test implementation consistency across eight mutation adapters; they are not 96 independent task families.

Executor	Schema	Raw unique	Raw ambiguous	TG unique	TG ambiguous
Git	append file	12/12	12/12 wrong	12/12	12/12 abstain
Git	replace file	12/12	12/12 wrong	12/12	12/12 abstain
Git	delete file	12/12	12/12 wrong	12/12	12/12 abstain
Git	rename file	12/12	12/12 wrong	12/12	12/12 abstain
SQLite	close ticket	12/12	12/12 wrong	12/12	12/12 abstain
SQLite	cancel order	12/12	12/12 wrong	12/12	12/12 abstain
SQLite	assign contact	12/12	12/12 wrong	12/12	12/12 abstain
SQLite	archive document	12/12	12/12 wrong	12/12	12/12 abstain

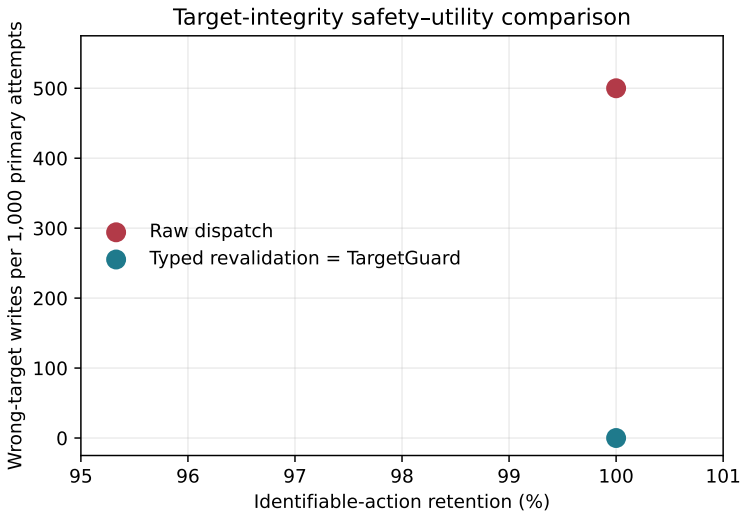


Figure 1: Primary safety–utility comparison from the same 192-case Git+SQLite executor result file used in Table 1. The x-axis is retention on uniquely identifiable rows and the y-axis is observed wrong-target writes per 1,000 attempts. All policies retain the identifiable work; typed revalidation and TargetGuard overlap at zero observed wrong-target writes. The stale 96-case stratum is reported separately as a freshness audit.

successful raw dispatches into the wrong entity. The auxiliary 1,800-case suite supplies the broader analytical policy comparison in the supplement: its scenario labels move from 375 to 0 per 1,000 for raw alias and TargetGuard, but those labels are not executor-observed writes. In the separate stale stratum, both guarded policies reject 96/96 stale actions. Each records zero collateral mutations, zero runtime errors, and 288/288 recorded audit passes, with no model calls. Their outcome fields match on all 288 paired case keys. These fixed-composition rates compare policies; they are not deployment prevalence estimates.

Strong-reference result. The typed revalidation policy performs its own typed lookup and pre-mutation re-read; it does not call TargetGuard's resolver. It matches TargetGuard on every native executor case: 96 intended commits, 96 ambiguity abstentions, 96 stale-action rejections, and no wrong-target, collateral, or runtime failures. On these cases, the certificate does not improve safety or retained work over a competent revalidation monitor. The evidence supports a common measurable boundary and a portable certificate/audit interface, not general equivalence or algorithmic dominance. Because the original and add-on runners expose different timing boundaries, we make no relative cost claim.

What the comparison establishes. The native comparison shows the same decision boundary in two executor mechanisms and eight mutation schemas. Application replays test two further adapters; the resolver-only boundary audit tests missing and malformed relations without invoking an executor. Together they demonstrate how a shared contract can organize distinct measurements. Matching the strong reference establishes agreement on these cases, not general algorithmic equivalence or a performance advantage.

In D12 the model response is held fixed: all 12 repair-off arms write to the constructed wrong target, and all 12 repair-on arms write to the designated target. Thus $\widehat{\Delta}_{\mathrm{pair}} = 1.00$ on these fixed pairs. Every arm records one mutation and no collateral effect. This isolates the effect of target substitution after designation, not the ability to infer the target. Shared templates and a fixed model/snapshot preclude treating these pairs as independent deployment samples. D12 contains collision-only interventions and does not identify false rejection on clean or ambiguous inputs; its role is a mechanism check, not an agent success-rate estimate.

D12 audit contract. Each causal row required a pre-fixed collision, identical paired response hashes, exactly one designated mutation with no collateral mutation, and retention of model, parse, grounding, execution, rejection, and cost fields. The full checklist and raw audit are supplied in the supplement.

Model interface boundary. The three-model runs test whether model outputs can enter the same certified executor path; they do not define the algorithm's safety guarantee. Qwen3.6:27B completed the preregistered atomic protocol, whereas Llama-3.1:8B and DeepSeek-R1:7B exposed retained grounding or serialization failures. On the held-out AppWorld Filesystem adapter, Qwen3.6:27B and Llama-3.1:8B preserve clean mutations and safely abstain on non-identifiable rows; DeepSeek-R1:7B returns empty `no_think` responses. Complete per-model counts and failures remain in the supplement and are not pooled into the headline comparison.

Secondary sensitivity analysis. The executor comparison in Table 1 is the primary result. The supplementary artifact reports the separate 1,800-case analytical policy comparison and maps its frozen outcome counts to declared wrong-write, false-rejection, and review weights for deployment planning. Those values are benchmark calibrations, not measured prevalence, prices, throughput, or industrial savings; no calibrated scalar is used as the paper's headline result.

7 Reproducibility and conclusion

The supplementary artifact's `EVIDENCE_INDEX.md` maps the native comparison (E1), analytical suite (E2), application replays (E3), boundary test (E4), D12 (E5), and model compatibility (E6) to records and source snapshots. Its verification script checks hashes and recorded counts, not independent experimental replication. Historical source/run binding gaps and incomplete state retention are disclosed; failed pilots are preserved but never pooled. The central contribution is one measurement and execution boundary: the target-integrity contract records a uniquely verified commit or a pre-mutation abstention as a separate target-integrity outcome. In the primary Git/SQLite executor comparison, raw dispatch produces 500 wrong-target writes per 1,000 primary attempts. Both the typed revalidation reference and TargetGuard reduce that observed rate to zero while retaining all uniquely identifiable work, and both reject all 96 actions made stale before revalidation in the separate freshness stratum. Their fixed-case parity places the contribution where the evidence supports it: a conditional, portable benchmark and commit-time contract, not superiority over a competent reference monitor. The 1,800-case CPU suite is an auxiliary analytical policy comparison, not a persistent-write measurement. The result does not establish a universal resolver, natural deployment prevalence, or industrial ROI.

AI use. Generative AI tools assisted with literature retrieval and discovery, research ideation, protocol and implementation discussion, code debugging, interpretation, drafting, and language editing. The authors made all final scientific decisions and take responsibility for the reported experiments,

source attributions, and limitations. Artifact checks verify record integrity and counts; they are not independent replication.

REFERENCES

[1] Rahul Suresh Babu and Shashank Indukuri. Entity Binding Failures in Tool-Augmented Agents. arXiv preprint, 2026.
[2] Rahul Suresh Babu and Shashank Indukuri. Binding Drift in Multi-Step Tool-Augmented Agents. arXiv preprint, 2026.
[3] Harsh Trivedi et al. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. ACL, 2024.
[4] Bhaskar Gurram. Auditing Automated Evaluation, Error Propagation, and Runtime Mitigation in Tool-Using Language Agents. arXiv preprint, 2026.
[5] Aarya Doshi et al. Towards Verifiably Safe Tool Use for LLM Agents. arXiv preprint, 2026.
[6] Jiarui Lu et al. ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities. arXiv preprint, 2024.
[7] Shunyu Yao et al. τ-Bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. arXiv preprint, 2024.
[8] Weihuang Zheng et al. E-Bench: Benchmarking Multi-Step Tool-Use Agents in Real-World Product Scenarios. arXiv preprint, 2026.
[9] Strick Sheng et al. When Agents Overtrust Environmental Evidence: An Extensible Agentic Framework for Benchmarking Evidence-Grounding Defects in LLM Agents. arXiv preprint, 2026.